

Sesión 1: Procesos

Concurrencia

Ángel Herranz

2019-2020

Universidad Politécnica de Madrid

En capítulos anteriores...

¿Y si pudiéramos hacer que los dos programas compartieran la misma *x*?

```
public static int x = 0;
```

```
public class Inc {  
    public static void  
        main(String args[]) {  
        x = x + 1;  
    }  
}
```

```
public class Dec {  
    public static void  
        main(String args[]) {  
        x = x - 1;  
    }  
}
```

Simultaneidad
+
Interacción

Simultaneidad
+
Sincronización y Comunicación

Simultaneidad

+

Sincronización y Comunicación

Simultaneidad

- ¿Cómo podemos lanzar *dos o más main a la vez*?
- Cada lenguaje de programación tiene sus formas
- Java tiene dos formas primitivas:
 - Subclase de la clase **Thread**
 - Implementación de la interfaz **Runnable**
- Hay más formas: *thread pools* (por ejemplo `ExecutorService` / `parallelStream`)

¹En *la industria* se usa menos ya que es de *bajo nivel*.

Simultaneidad

- ¿Cómo podemos lanzar **dos o más *main* a la vez?**
- **Cada lenguaje** de programación tiene sus **formas**
- Java tiene dos formas **primitivas**:
 - **Subclase de la clase *Thread***
 - Implementación de la interfaz ***Runnable***
- Hay más formas: *thread pools* (por ejemplo `ExecutorService` / `parallelStream`)
- Nosotros vamos a usar la clase ***Thread***¹

¹En *la industria* se usa menos ya que es de *bajo nivel*.

Lesson: Concurrency

Leer antes de la primera entrega

Lesson: Concurrency

En particular

Threads Objects

Mundos paralelos

Escribe, compila y ejecuta **varias veces**

```
1 public class HolaMundos {
2     private static class HolaMundo
3         extends Thread {
4         public HolaMundo() {
5         }
6         public void run() {
7             System.out.println(
8                 "Hola mundo"
9             );
10        }
11    }
```

Herranz

```
13 public static void
14     main(String[] args) {
15         HolaMundo hola1 =
16             new HolaMundo();
17         HolaMundo hola2 =
18             new HolaMundo();
19         hola1.start();
20         hola2.start();
21         System.out.println(
22             "Hola, soy 'el main'"
23         );
24     }
25 }
```

Atentos

- Heredar de **Thread**
- **run** es el *nuevo* main
- **new** para crear los procesos
- ¿Entonces? ¿Donde está la *magia*?

Atentos

- Heredar de **Thread**
- **run** es el *nuevo* main
- **new** para crear los procesos
- ¿Entonces? ¿Donde está la *magia*?

start

⚠ Atentos

- Heredar de **Thread**
- **run** es el *nuevo main*
- **new** para crear los procesos
- ¿Entonces? ¿Donde está la *magia*?

start

- ¿Cuántos procesos hay?
- ¿Ves los *entrelazados*?
- Terminología: *proceso padre*

Leer en las dos primera semanas
Concepts and Notations for Concurrent
Programming. G.R. Andrews, F.B. Schneider
(1983 🖐️). Seccciones 1, 2, 3.1 y 3.2

Leer en las dos primera semanas
Concepts and Notations for Concurrent
Programming. G.R. Andrews, F.B. Schneider
(1983 🖐). Seccciones 1, 2, 3.1 y 3.2

Yo no voy a ser capaz de enseñaros más

Progress Finite Assumption

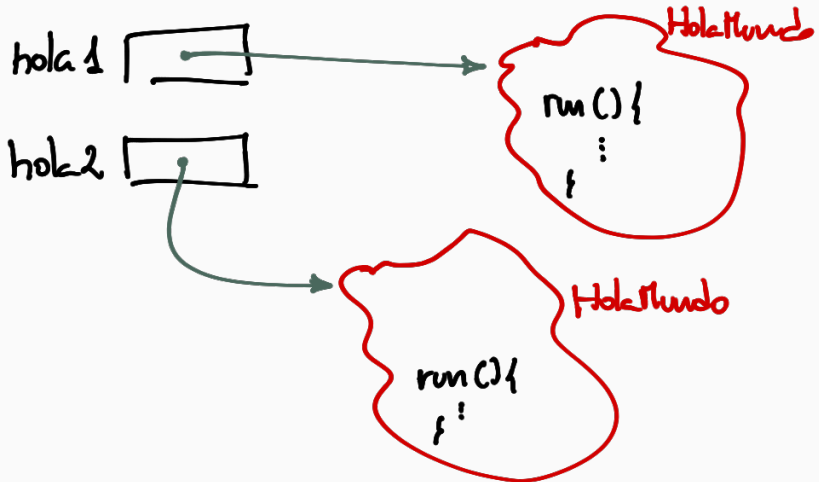
Única suposición sobre
velocidades de procesos:

Progress Finite Assumption

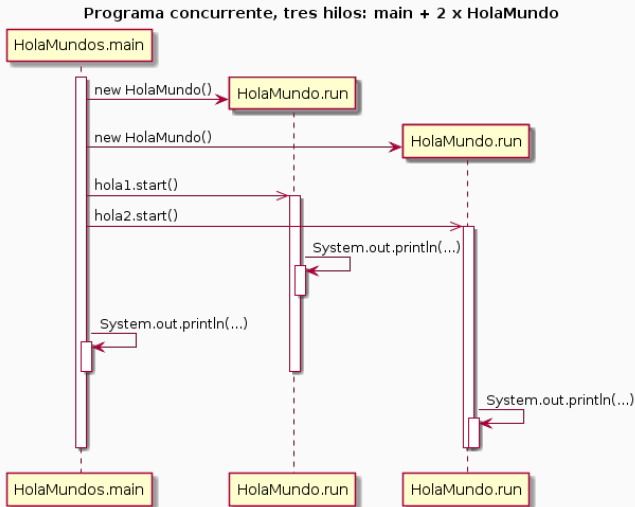
Única suposición sobre
velocidades de procesos:

La velocidad de los procesos es
finita y mayor de 0

Memoria



Escenario: ejemplo de *entrelazado*



Mundos paralelos 1 y 2

Modifica el programa para que cada proceso se identifique al escribir “Hola mundo” con un número diferente (ej. “Hola mundo 1” y “Hola mundo 2”)

¡Dibuja la memoria!

Join

- Igual de importante que lanzar un proceso:
`t.start()`
- es esperar a que un proceso termine
- Sea t un proceso, la semántica de

`t.join()`

es

“esperar a que el proceso t termine,
si ha terminado ya no hace nada”

Join

- Igual de importante que lanzar un proceso:
`t.start()`
- es esperar a que un proceso termine
- Sea t un proceso, la semántica de

`t.join()`

es

“esperar a que el proceso t termine,
si ha terminado ya no hace nada”

- Es nuestra primera directiva de sincronización



El *main* espera por sus *hijos*

¿Qué conseguimos con este código?

```
public static void main(String[] args) {  
    HolaMundo hola1 = new HolaMundo(1);  
    HolaMundo hola2 = new HolaMundo(2);  
    hola1.start(); hola2.start();  
    hola1.join(); hola2.join();  
    System.out.println("Hola, soy 'el main'");  
}
```

El *main* espera por sus *hijos*

¿Qué conseguimos con este código?

El *main* va a esperar a que terminen sus *hijos* antes de decir *Hola, soy el 'main'*

```
public static void main(String[] args) {  
    HolaMundo hola1 = new HolaMundo(1);  
    HolaMundo hola2 = new HolaMundo(2);  
    hola1.start(); hola2.start();  
    hola1.join(); hola2.join();  
    System.out.println("Hola, soy 'el main'");  
}
```

El *main* espera por sus *hijos*

¿Qué conseguimos con este código?

El *main* va a **esperar** a que terminen sus *hijos* antes de decir *Hola, soy el 'main'*

```
public static void main(String[] args) {  
    HolaMundo hola1 = new HolaMundo(1);  
    HolaMundo hola2 = new HolaMundo(2);  
    hola1.start(); hola2.start();  
    hola1.join(); hola2.join();  
    System.out.println("Hola, soy 'el main'");  
}
```


Pequeño problema técnico

error: unreported exception

*InterruptedException; must be caught or
declared to be thrown*

```
hola1.join();
```

^

Pequeño problema técnico

error: unreported exception

*InterruptedException; must be caught or
declared to be thrown*

```
hola1.join();
```

^

¿Opciones para tratarlo?



Ejercicio obligatorio semanal

Hoja de ejercicios en:

<http://babel.ls.fi.upm.es/teaching/concurrencia>

Ejercicio 1:

Creación de procesos en Java

Fichero a entregar:

`CC_01_Threads.java`

Sistema de entrega:

por anunciar